**Week 3 Hands-On Assignment**

**Exercise 1: Configuring a Basic Spring Application**

**Scenario:** Your company is developing a web application for managing a library. You need to use the Spring Framework to handle the backend operations.

**Steps:**

1. **Set Up a Spring Project:**
   * Create a Maven project named LibraryManagement.
   * Add Spring Core dependencies in the pom.xml file.
2. **Configure the Application Context:**
   * Create an XML configuration file named applicationContext.xml in the src/main/resources directory.
   * Define beans for BookService and BookRepository in the XML file.
3. **Define Service and Repository Classes:**
   * Create a package com.library.service and add a class BookService.
   * Create a package com.library.repository and add a class BookRepository.
4. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

**Exercise 2: Implementing Dependency Injection**

**Scenario:** In the library management application, you need to manage the dependencies between the BookService and BookRepository classes using Spring’s IoC and DI.

**Steps:**

1. **Modify the XML Configuration:**
   * Update applicationContext.xml to wire BookRepository into BookService.
2. **Update the BookService Class:**
   * Ensure that BookService class has a setter method for BookRepository.
3. **Test the Configuration:**
   * Run the LibraryManagementApplication main class to verify the dependency injection.

**Exercise 4: Creating and Configuring a Maven Project**

**Scenario:** You need to set up a new Maven project for the library management application and add Spring dependencies.

**Steps:**

1. **Create a New Maven Project:**
   * Create a new Maven project named LibraryManagement.
2. **Add Spring Dependencies in** ``**:**
   * Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.
3. **Configure Maven Plugins:**
   * Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

**Hands-On 1: Spring Data JPA - Quick Example**

**Software Pre-requisites:**

* MySQL Server 8.0
* MySQL Workbench 8
* Eclipse IDE for Enterprise Java Developers 2019-03 R
* Maven 3.6.2

**Steps:**

1. **Create a Spring Boot Project:**
   * Go to <https://start.spring.io/>
   * Group: com.cognizant
   * Artifact Id: orm-learn
   * Description: Demo project for Spring Data JPA and Hibernate
   * Add Dependencies: Spring Boot DevTools, Spring Data JPA, MySQL Driver
   * Generate the project zip, extract to Eclipse Workspace, and import as Maven project.
2. **Create Schema in MySQL:**

mysql -u root -p  
mysql> create schema ormlearn;

1. **Configure** ``**:**

logging.level.org.springframework=info  
logging.level.com.cognizant=debug  
logging.level.org.hibernate.SQL=trace  
logging.level.org.hibernate.type.descriptor.sql=trace  
logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger{25} %25M %4L %m%n  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn  
spring.datasource.username=root  
spring.datasource.password=root  
spring.jpa.hibernate.ddl-auto=validate  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

1. **Build and Run Project:**

mvn clean package

1. **Main Class Logging:**

import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);  
  
public static void main(String[] args) {  
 SpringApplication.run(OrmLearnApplication.class, args);  
 LOGGER.info("Inside main");  
}

**Project Structure Overview:**

1. src/main/java - Application source code
2. src/main/resources - Application configuration
3. src/test/java - Unit and integration tests
4. OrmLearnApplication.java - Contains main() method with @SpringBootApplication
5. pom.xml - Maven project dependencies and plugins

**Country Table Creation:**

create table country(co\_code varchar(2) primary key, co\_name varchar(50));  
insert into country values ('IN', 'India');  
insert into country values ('US', 'United States of America');

**Persistence Class:**

@Entity  
@Table(name="country")  
public class Country {  
  
 @Id  
 @Column(name="code")  
 private String code;  
  
 @Column(name="name")  
 private String name;  
  
 // Getters, Setters, toString  
}

**Repository Interface:**

@Repository  
public interface CountryRepository extends JpaRepository<Country, String> {  
}

**Service Class:**

@Service  
public class CountryService {  
  
 @Autowired  
 private CountryRepository countryRepository;  
  
 @Transactional  
 public List<Country> getAllCountries() {  
 return countryRepository.findAll();  
 }  
}

**Testing in OrmLearnApplication.java:**

private static CountryService countryService;  
  
private static void testGetAllCountries() {  
 LOGGER.info("Start");  
 List<Country> countries = countryService.getAllCountries();  
 LOGGER.debug("countries={}", countries);  
 LOGGER.info("End");  
}  
  
public static void main(String[] args) {  
 ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);  
 countryService = context.getBean(CountryService.class);  
 testGetAllCountries();  
}

**Output:**

Inside main  
Start  
countries=[Country{code='IN', name='India'}, Country{code='US', name='United States of America'}]  
End

**JPA vs Hibernate vs Spring Data JPA**

| Feature | JPA | Hibernate | Spring Data JPA |
| --- | --- | --- | --- |
| Type | Specification | Implementation of JPA | Abstraction over JPA |
| Boilerplate | High | Moderate | Very low (reduced) |
| Transaction Mgmt | Manual | Manual or Spring Integrated | Managed by Spring |
| Code Simplicity | Complex | Medium | Simple and clean |

**Reference Links:**

* [Java World: What is JPA](https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html)
* [DZone: Hibernate vs Spring Data JPA](https://dzone.com/articles/what-is-the-difference-between-hibernate-and-sprin-1)